

# DW\_fifoctrl\_s1\_sf

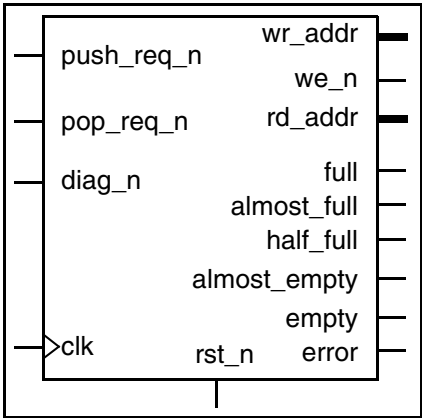
## Synchronous (Single-Clock) FIFO Controller with Static Flags

Version, STAR, and myDesignWare Subscriptions: [IP Directory](#)

### Features and Benefits

- Fully registered synchronous address and flag output ports
- All operations execute in a single clock cycle
- FIFO empty, half full, and full flags
- FIFO error flag indicating underflow, overflow, and pointer corruption
- Parameterized word depth
- Parameterized almost full and almost empty flags
- Parameterized reset mode (synchronous or asynchronous)
- Interfaces to common hard macro or compiled ASIC dual-port synchronous RAMs
- Includes a low-power implementation that has power benefits from minPower optimization (for details, see [Table 1-3](#) on page 3)

### Revision History



### Description

DW\_fifoctrl\_s1\_sf is a FIFO RAM controller designed to interface with a dual-port synchronous RAM with optional low power benefits.

The RAM must have:

- A synchronous write port, and
- Either an asynchronous or synchronous read port.

The FIFO controller provides address generation, write-enable logic, flag logic, and operational error detection logic. Parameterizable features include FIFO depth (up to 24 address bits or 16,777,216 locations), almost empty level, almost full level, level of error detection, and type of reset (either asynchronous or synchronous). You specify these parameters when the controller is instantiated in the design.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Reset input, active low <ul style="list-style-type: none"><li>■ Asynchronous if <i>rst_mode</i> = 0</li><li>■ Synchronous if <i>rst_mode</i> = 1</li></ul>

**Table 1-1 Pin Description (Continued)**

Pin Name	Width	Direction	Function
push_req_n	1 bit	Input	FIFO push request, active low
pop_req_n	1 bit	Input	FIFO pop request, active low
diag_n	1 bit	Input	Diagnostic control For <i>err_mode</i> = 0, NC For other <i>err_mode</i> values), active low
we_n	1 bit	Output	Write enable output for write port of RAM, active low
empty	1 bit	Output	FIFO empty output, active high
almost_empty	1 bit	Output	FIFO almost empty output, asserted when FIFO level $\leq$ <i>ae_level</i> , active high
half_full	1 bit	Output	FIFO half full output, active high
almost_full	1 bit	Output	FIFO almost full output, asserted when FIFO level $\geq$ ( <i>depth</i> - <i>af_level</i> ); active high
full	1 bit	Output	FIFO full output, active high
error	1 bit	Output	FIFO error output, active high
wr_addr	$\text{ceil}(\log_2[\text{depth}])$ bits	Output	Address output to write port of RAM
rd_addr	$\text{ceil}(\log_2[\text{depth}])$ bits	Output	Address output to read port of RAM

**Table 1-2 Parameter Description**

Parameter	Values	Function
depth	2 to $2^{24}$ Default: 4	Number of memory elements used in FIFO (used to size the address ports)
ae_level	1 to $depth - 1$ Default: 1	Almost empty level (the number of words in the FIFO at or below which the <code>almost_empty</code> flag is active)
af_level	1 to $depth - 1$ Default: 1	Almost full level (the number of empty memory locations in the FIFO at which the <code>almost_full</code> flag is active. For more about status flags, see <a href="#">Figure 1-2</a> on page 8.)
err_mode	0 to 2 Default: 0	Error mode <ul style="list-style-type: none"> <li>0: Underflow/overflow and pointer latched checking</li> <li>1: Underflow/overflow latched checking,</li> <li>2: Underflow/overflow unlatched checking</li> </ul>
rst_mode	0 or 1 Default: 0	Reset mode <ul style="list-style-type: none"> <li>0: Asynchronous reset</li> <li>1: Synchronous reset</li> </ul>

**Table 1-3 Synthesis Implementations**

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare
lpwr <sup>a</sup>	Low Power synthesis model	<ul style="list-style-type: none"> <li>DesignWare (P-2019.03 and later)</li> <li>DesignWare-LP (before P-2019.03)</li> </ul>

a. Requires that you enable minPower; for details, see [“Enabling minPower”](#) on page 16.  
When minPower is enabled, the lpwr implementation is always chosen during synthesis.

**Table 1-4 Simulation Models**

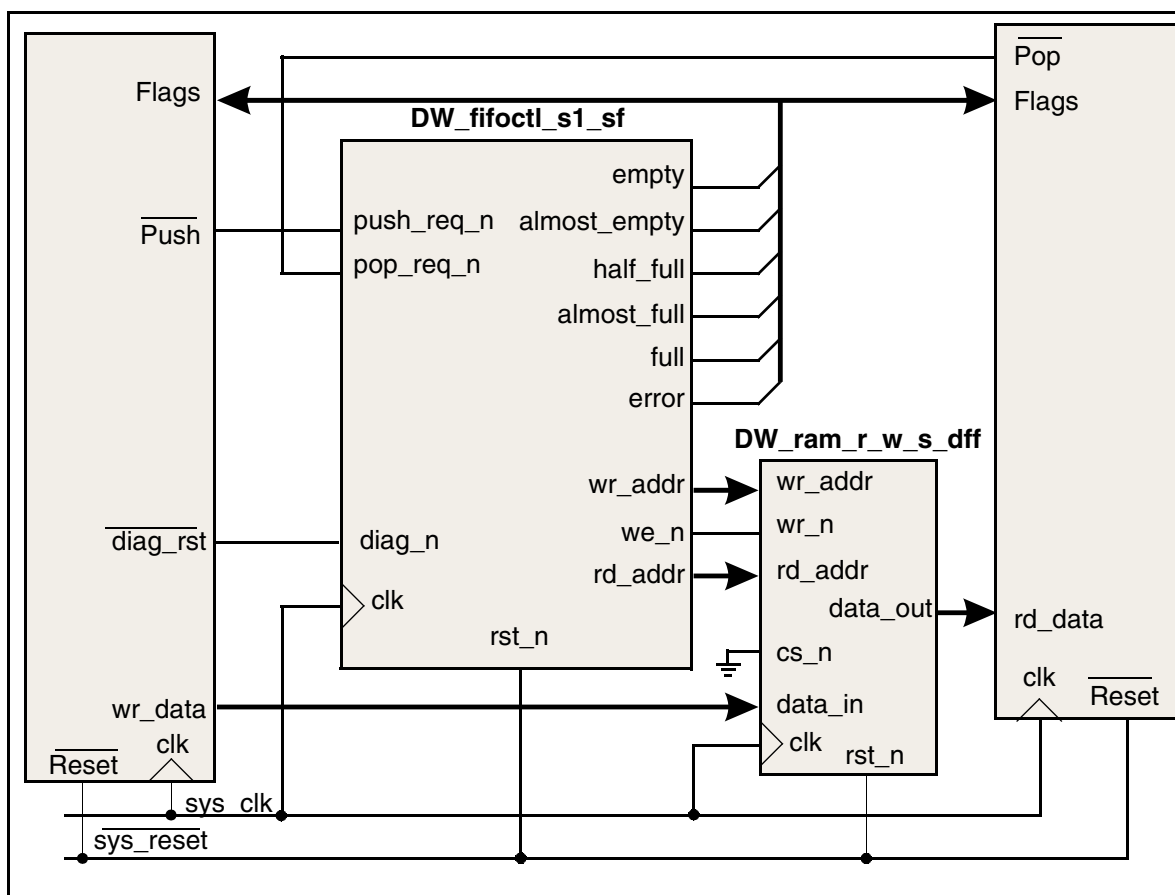
Model	Function
DW03.DW_FIFOCTL_S1_SF_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_fifoctl_s1_sf_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fifoctl_s1_sf.v	Verilog simulation model source code

**Table 1-5 Error Mode Description**

err_mode	Error Types Detected	Error Output	diag_n
0	Underflow/Overflow and Pointer Corruption	Latched	Connected
1	Underflow/Overflow	Latched	N/C
2	Underflow/Overflow	Not Latched	N/C

Figure 1-1 shows a typical application of the controller.

**Figure 1-1 Example Usage of DW\_fifoctrl\_s1\_sf**



## Writing to the FIFO (Push)

The `wr_addr` and `we_n` output ports of the FIFO controller provide the write address and synchronous write enable to the FIFO.

A push is executed when the `push_req_n` input is asserted (low), and either:

- The `full` flag is inactive (low),

or:

- The `full` flag is active (high), and
- The `pop_req_n` input is asserted (low).

Thus, a push can occur even if the FIFO is full as long as a pop is executed in the same cycle.

Asserting `push_req_n` in either of the above cases causes the following events to occur:

- The `we_n` is asserted immediately, preparing for a write to the RAM on the next clock, and
- On the next rising edge of `clk`, `wr_addr` is incremented.

Thus, the RAM is written, and `wr_addr` (which always points to the address of the next word to be pushed) is incremented on the same rising edge of `clk`—the first clock after `push_req_n` is asserted. This means that `push_req_n` must be asserted early enough to propagate through the FIFO controller to the RAM before the next clock.

## Write Error

An error occurs if a push is attempted while the FIFO is full. That is, if:

- The `push_req_n` input is asserted (low),
- The `full` flag is active (high), and
- The `pop_req_n` input is inactive (high).

## Reading from the FIFO (Pop)

The read port of the RAM can be either synchronous or asynchronous. In either case, the `rd_addr` output port of the DW\_ffioctl\_s1\_sf provides the read address to the RAM. The `rd_addr` output bus always points to, thus prefetches, the next word of RAM read data to be popped.

A pop operation occurs when `pop_req_n` is asserted (low), as long as the FIFO is not empty. Asserting `pop_req_n` causes the `rd_addr` pointer to be incremented on the next rising edge of `clk`. Thus, the RAM read data must be captured on the `clk` following the assertion of `pop_req_n`. For RAMs with a synchronous read port, the output data is captured in the output stage of the RAM. For RAMs with an asynchronous read port, the output data is captured by the next stage of logic after the FIFO.

Refer to the timing diagrams for details of the pop operation for RAMs with synchronous and asynchronous read ports.

## Read Error

An error occurs if:

- The `pop_req_n` input is active (low), and
- The empty flag is active (high).

## Simultaneous Push and Pop

Push and pop can occur at the same time if there is data in the FIFO, even when the FIFO is full. With the FIFO not empty, `rd_addr` is pointing to the next address to be popped, and the pop data is available to be prefetched at the RAM output.

When `pop_req_n` and `push_req_n` are both asserted, the following events occur on the next rising edge of `clk`:

- Pop data is captured by the next stage of logic after the FIFO, and
- The new data is pushed into the same location from which the data was popped.

Thus, there is no conflict in a simultaneous push and pop when the FIFO is full. A simultaneous push and pop cannot occur when the FIFO is empty, since there is no pop data to prefetch.

## Reset

### **rst\_mode**

This parameter selects whether reset is asynchronous (*rst\_mode* = 0) or synchronous (*rst\_mode* = 1). If asynchronous mode is selected, asserting `rst_n` (setting it low) immediately causes the internal address pointers to be set to 0, and the flags and error outputs to be initialized. If synchronous mode is selected, the address pointers, flags, and error outputs are initialized at the rising edge of `clk` after `rst_n` is asserted.

The error outputs and flags are initialized as follows:

- The empty and almost\_empty are initialized to 1, and
- All other flags and the error output are initialized to 0.

## Errors

### **err\_mode**

The *err\_mode* parameter determines which possible fault conditions are detected, and whether the error output remains active until reset or for only the clock cycle in which the error was detected.

When the *err\_mode* parameter is set to 0 at design time, the `diag_n` input provides an unconditional synchronous reset to the value of the `rd_addr` output port. This can be used to intentionally cause the FIFO address pointers to become corrupted, forcing a pointer inconsistency-type error.

For normal operation when *err\_mode* = 0, `diag_n` should be driven inactive (high). When *err\_mode* is set to 1 or 2, the `diag_n` input is ignored (unconnected).

### **error**

The `error` output indicates a fault in the operation of the FIFO control logic. There are several possible causes for the error output to be activated:

1. Overflow (push and no pop while full).
2. Underflow (pop while empty).
3. Empty pointer mismatch ( $rd\_addr \neq wr\_addr$  when empty).
4. Full pointer mismatch ( $rd\_addr \neq wr\_addr$  when full).
5. In between pointer mismatch ( $rd\_addr = wr\_addr$  when neither empty nor full).

When *err\_mode* = 0, all five causes are detected, and the *error* output (once activated) remains active until reset. When *err\_mode* = 1, only causes 1 and 2 are detected, and the *error* output (once activated) remains active until reset. When *err\_mode* = 2, only causes 1 and 2 are detected, and the *error* output only stays active for the clock cycle in which the *error* is detected. For error mode descriptions, see [Table 1-5](#) on page 4. The *error* output is set low when *rst\_n* is applied.

## Controller Status Flag Outputs

Refer to [Figure 1-2](#) on page 8 for operation of the status flags.

### **empty**

The *empty* output indicates that there are no words in the FIFO available to be popped. The *empty* output is set high when *rst\_n* is applied.

### **almost\_empty**

The *almost\_empty* output is asserted when there are no more than *ae\_level* words currently in the FIFO available to be popped. The *ae\_level* parameter defines the almost empty threshold. The *almost\_empty* output signal is useful for preventing the FIFO from underflowing. The *almost\_empty* output is set high when *rst\_n* is applied.

### **half\_full**

The *half\_full* output is active high when at least half the FIFO memory locations are occupied. The *half\_full* output is set low when *rst\_n* is applied.

### **almost\_full**

The *almost\_full* output is asserted when there are no more than *af\_level* empty locations in the FIFO. The *af\_level* parameter defines the almost full threshold. The *almost\_full* output signal is useful for preventing the FIFO from overflowing. The *almost\_full* output is set low when *rst\_n* is applied.

### **full**

The *full* output indicates that the FIFO is full and there is no space available for push data. The *full* output is set low when *rst\_n* is applied.

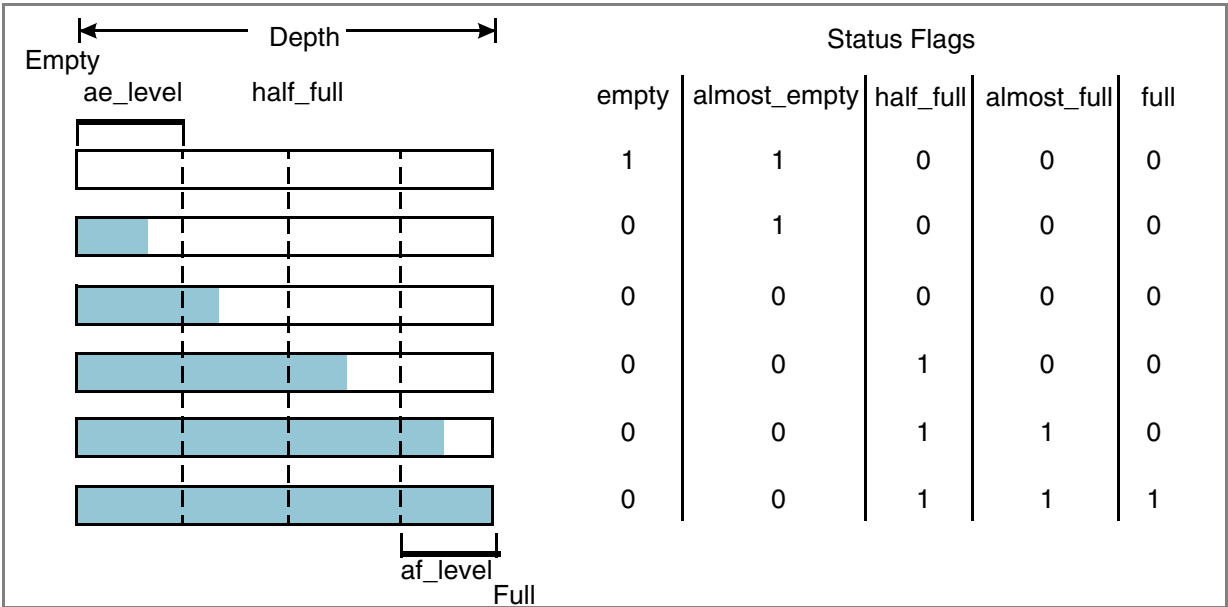
Application Notes

The *ae\_level* parameter value is chosen at design time to give the input flow control logic enough time to begin pushing data into the FIFO before the last word is popped by the output flow control logic.

The *af\_level* value is chosen at design time to give the output flow control logic enough time to begin popping data out of the FIFO. In other situations, this time is needed to allow the input flow control logic to interrupt the pushing of data into the FIFO.

Figure 1-2 shows the status flags of the DW\_fifoctl\_s1\_sf FIFO controller at various FIFO storage levels.

Figure 1-2 DW\_fifoctl\_s1\_sf FIFO Status Flags





## Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW\_SUPPRESS\_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

```
`define DW_SUPPRESS_WARN
```

- Or, include a command line option to the simulator, such as:

```
+define+DW_SUPPRESS_WARN (which is used for the Synopsys VCS simulator)
```

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

```
WARNING: <instance_path>.<clock_name>_monitor:  
at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
```

To suppress only this warning message for all DWBB components, use the following macro:

- Define the DW\_DISABLE\_CLK\_MONITOR macro. You can define this macro in the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

```
`define DW_DISABLE_CLK_MONITOR
```

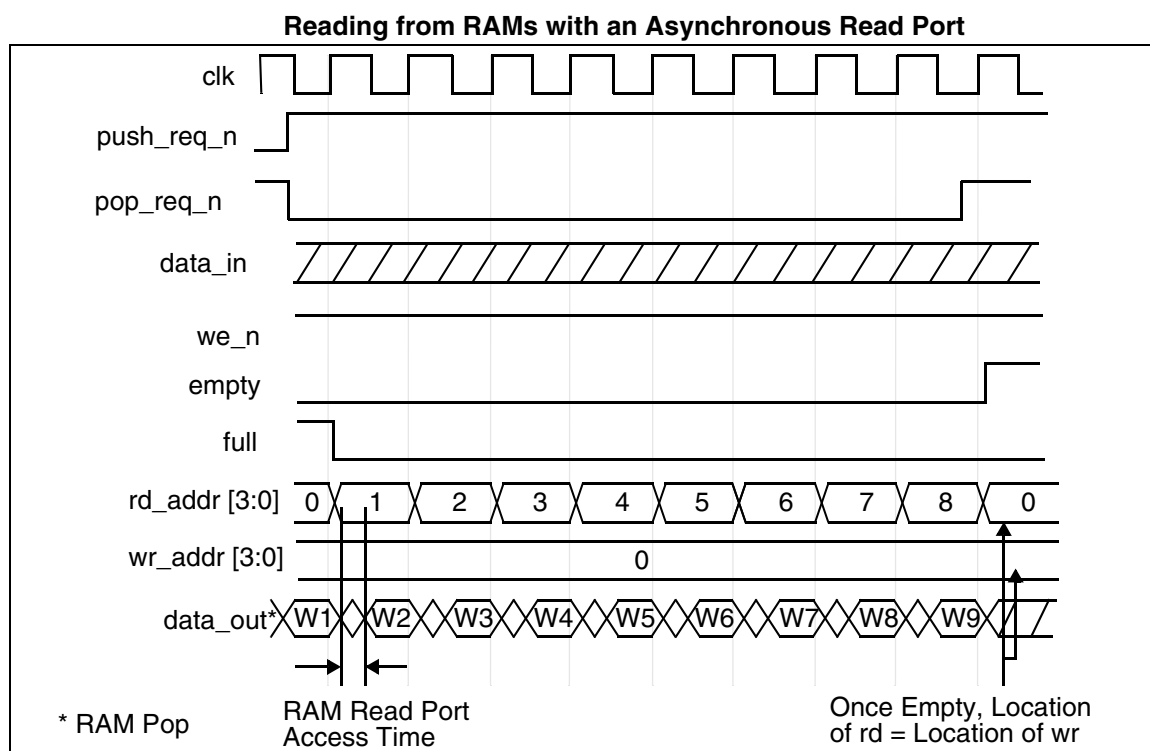
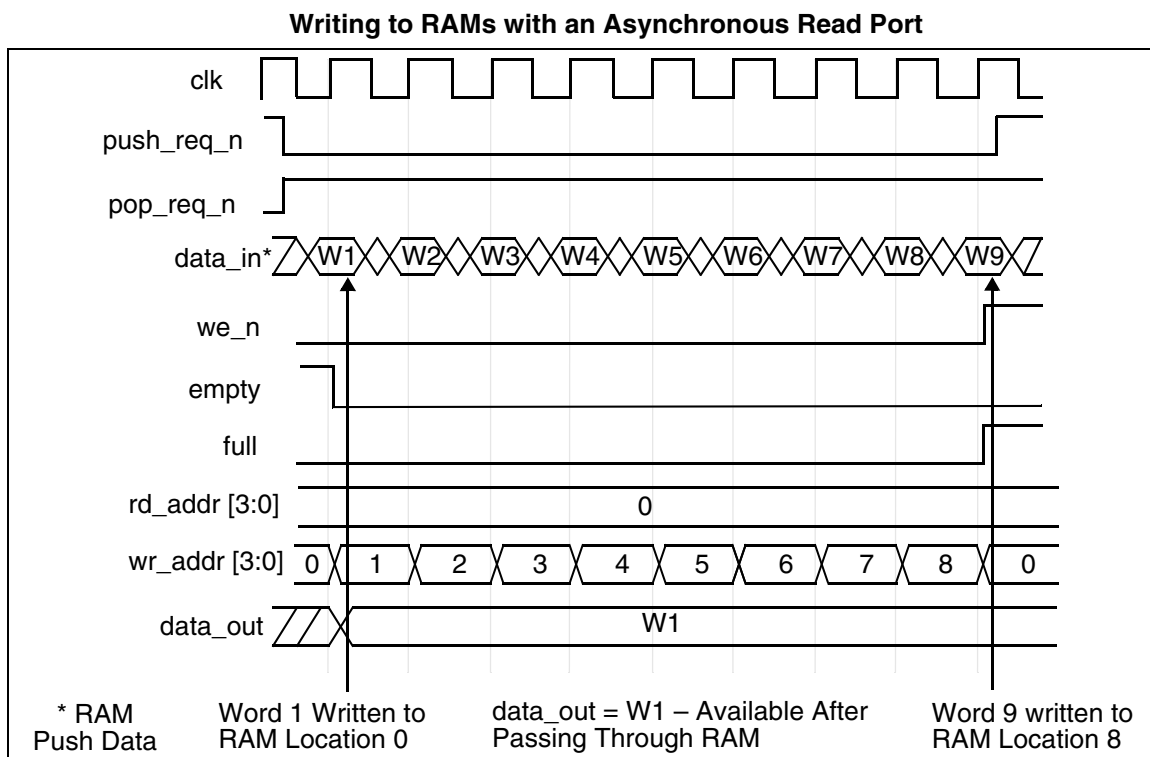
- Or, include a command line option to the simulator, such as:

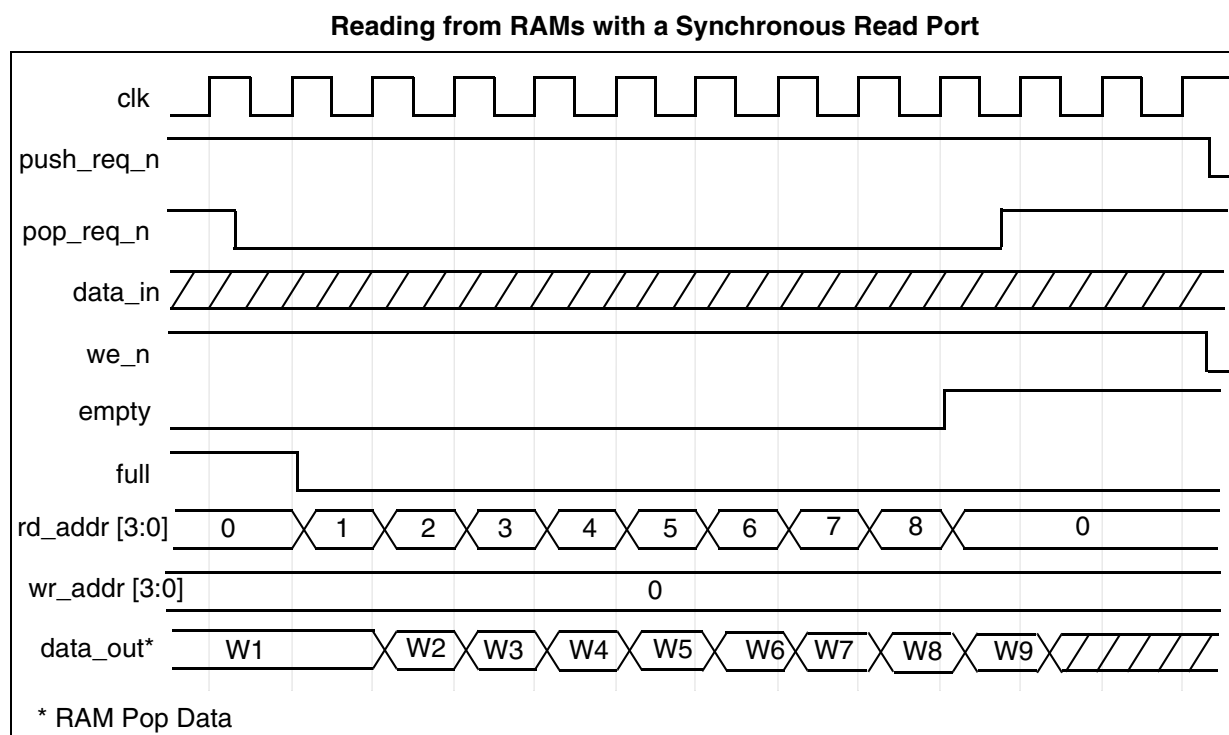
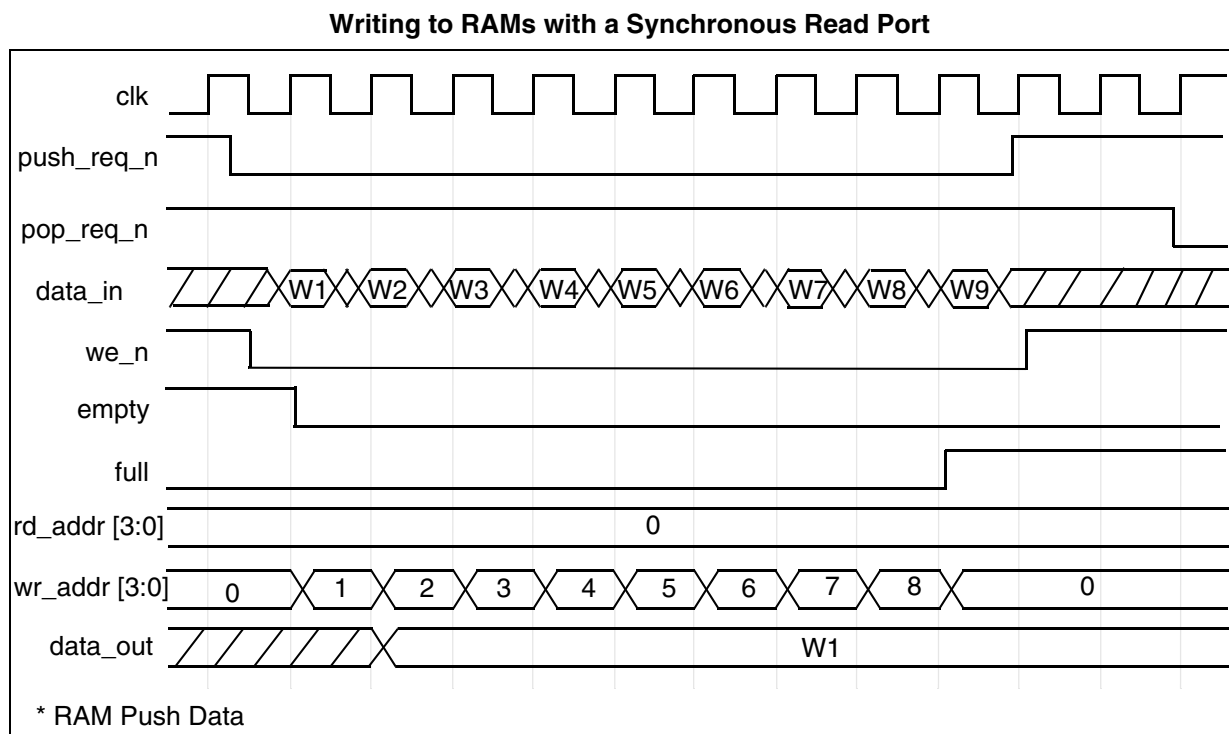
```
+define+DW_DISABLE_CLK_MONITOR (which is used for the Synopsys VCS simulator)
```

This message is also suppressed using the DW\_SUPPRESS\_WARN macro explained earlier.

## Timing Waveforms

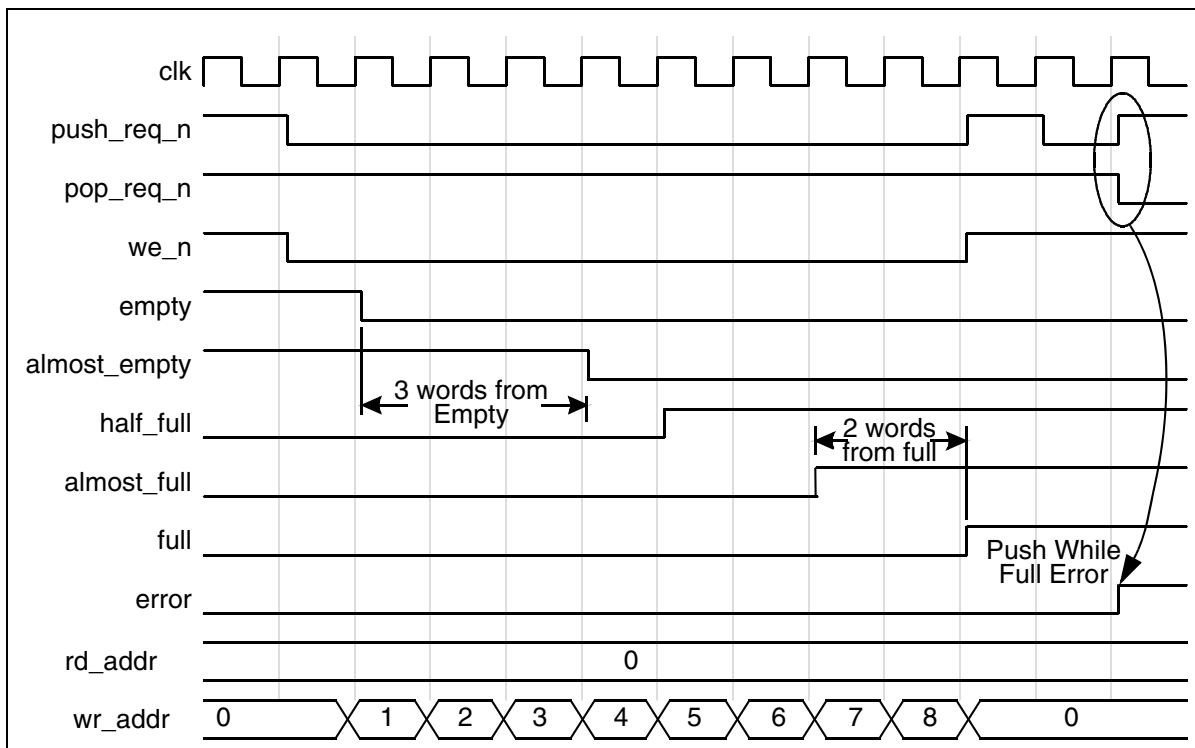
Figure 1-3 Push and Pop Timing Waveforms (Asynchronous Read Port RAMs)



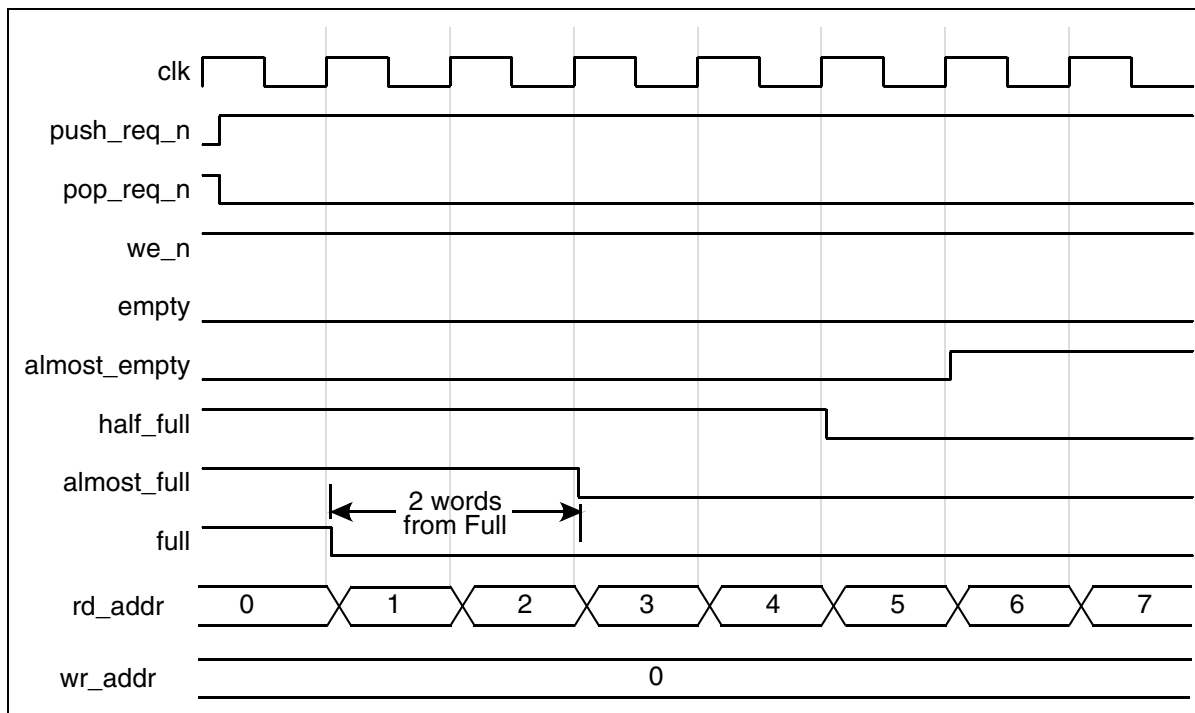
**Figure 1-4 Push and Pop Timing Waveforms (Synchronous Read Port RAMs)**

**Figure 1-5 Status Flag Timing Waveforms While Popping**

**Writing to RAMs Using DW\_fifoc1\_s1\_sf with depth = 9, ae\_level = 3, af\_level = 2**



**Reading from RAMs Using DW\_fifoc1\_s1\_sf with depth = 9, ae\_level = 3, af\_level = 2**



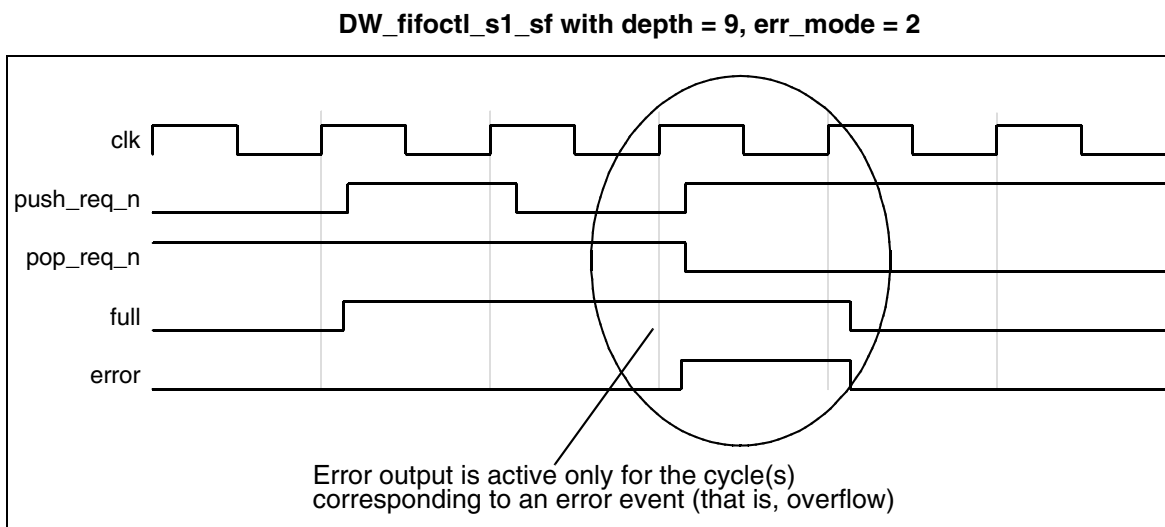
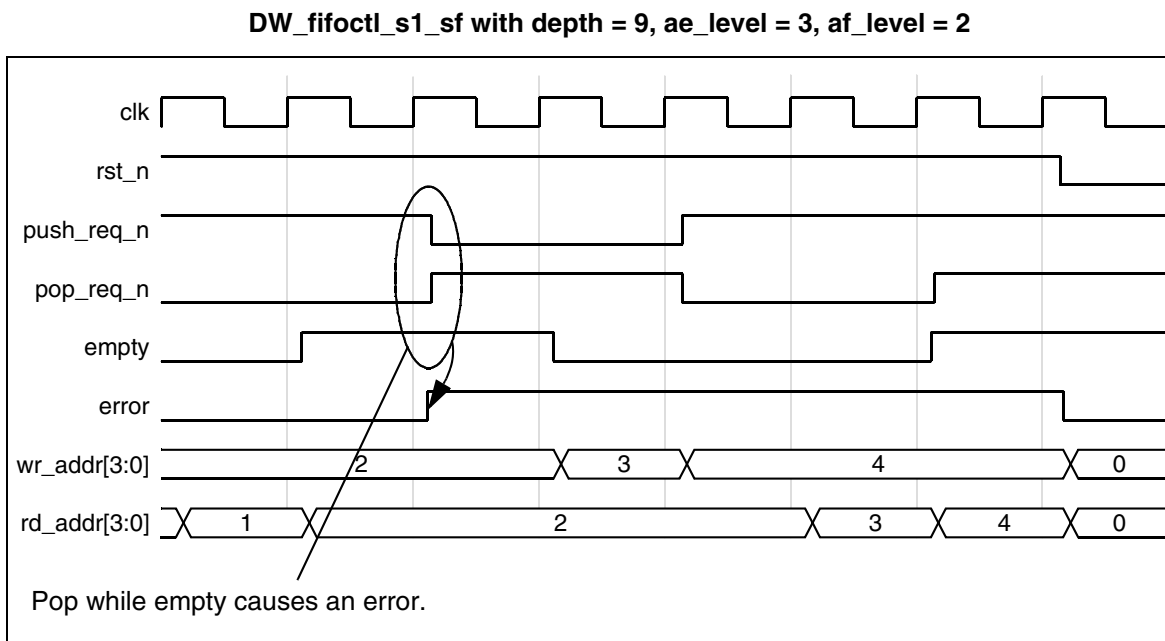
**Figure 1-6 Error Flag Timing Waveforms**

Figure 1-7 Error Flag Timing Waveforms (continued)

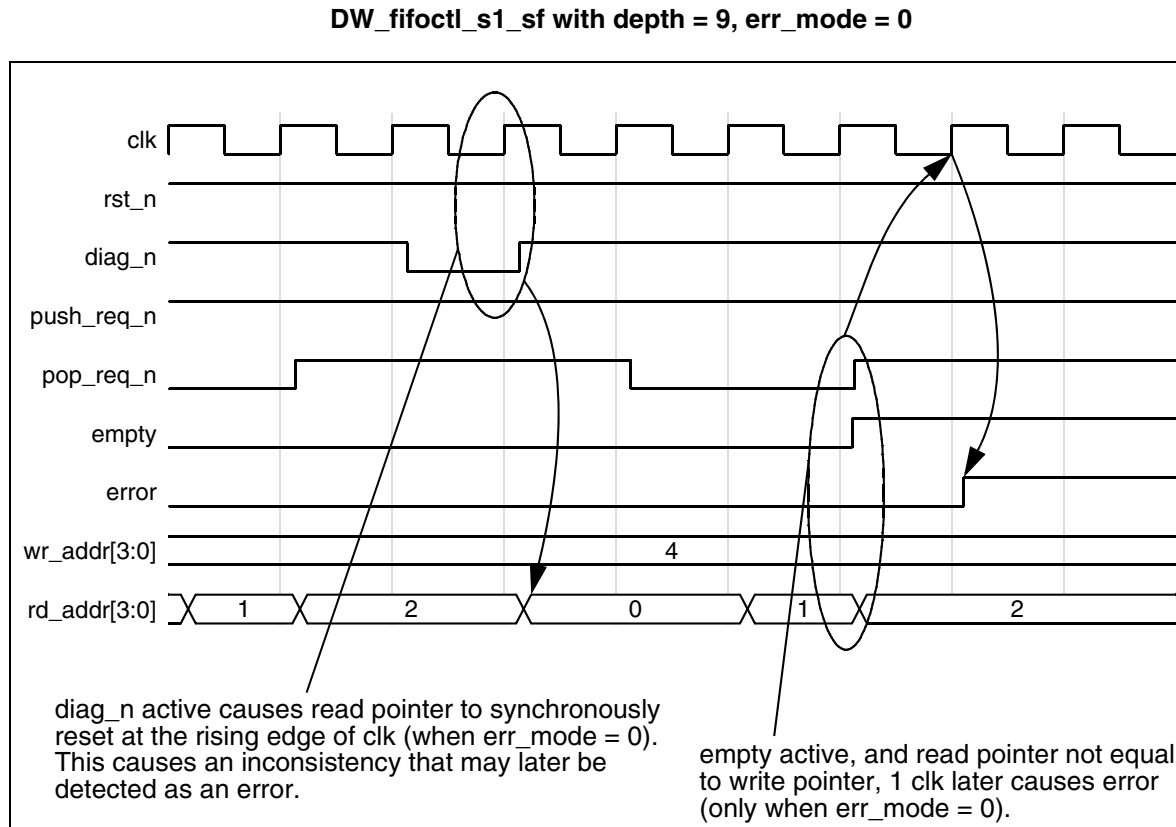


Figure 1-8 Error Flag Timing Waveforms (continued)

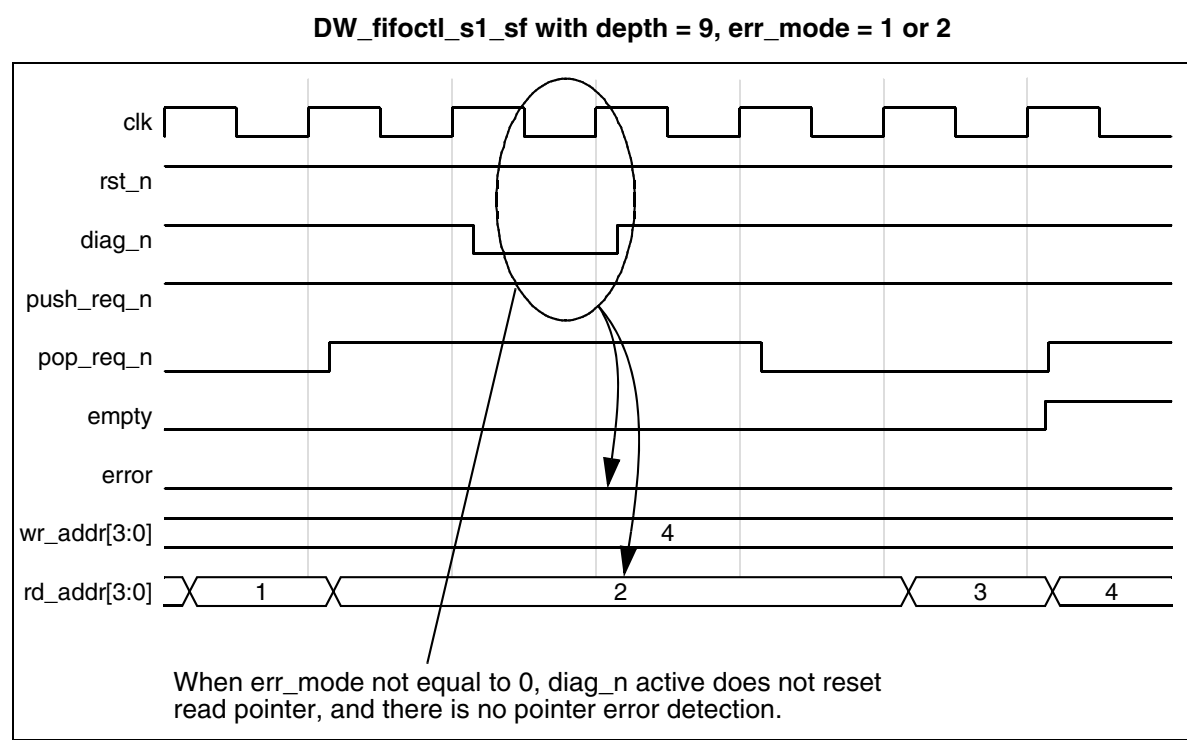
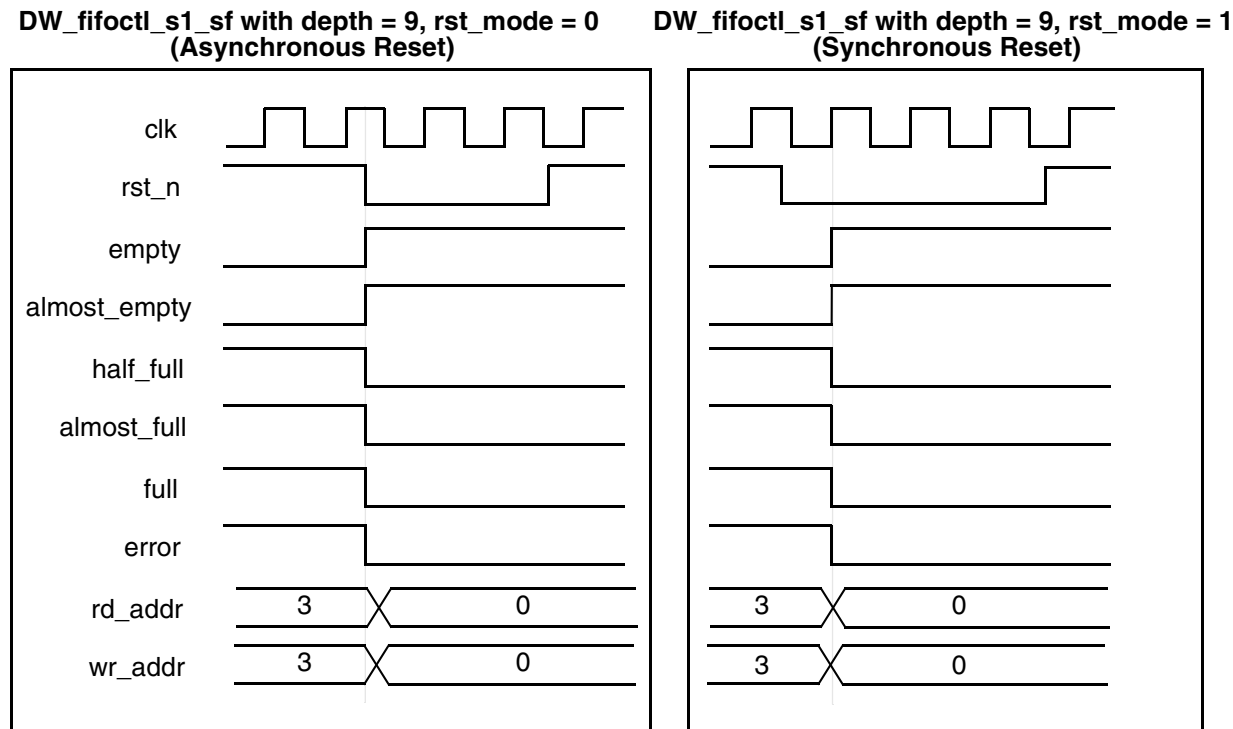


Figure 1-9 Reset Timing Waveforms



## Enabling minPower

You can instantiate this component without enabling minPower, but to achieve power savings from the low-power implementation “lpwr” (see [Table 1-3](#) on page 3), you must enable minPower optimization, as follows:

- Design Compiler

- Version P-2019.03 and later:

```
set power_enable_minpower true
```

- Before version P-2019.03 (requires the DesignWare-LP license feature):

```
set synthetic_library {dw_foundation.sldb dw_minpower.sldb}  
set link_library {* $target_library $synthetic_library}
```

- Fusion Compiler

Optimization for minPower is enabled as part of the total\_power metric setting. To enable the total\_power metric, use the following:

```
set_qor_strategy -stage synthesis -metric total_power
```

## Related Topics

- [Memory – FIFO Overview](#)
- [DesignWare Building Block IP User Guide](#)

## HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;  
use IEEE.std_logic_1164.all;  
use DWARE.DWpackages.all;  
use DWARE.DW_foundation_comp.all;  
  
entity DW_fifoctl_s1_sf_inst is  
  generic (inst_depth      : INTEGER := 4;  
           inst_ae_level   : INTEGER := 1;  
           inst_af_level   : INTEGER := 1;  
           inst_err_mode   : INTEGER := 0;  
           inst_rst_mode   : INTEGER := 0 );  
  port (inst_clk           : in std_logic;  
        inst_rst_n        : in std_logic;  
        inst_push_req_n   : in std_logic;  
        inst_pop_req_n    : in std_logic;  
        inst_diag_n       : in std_logic;  
        we_n_inst         : out std_logic;  
        empty_inst        : out std_logic;  
        almost_empty_inst : out std_logic;  
        half_full_inst    : out std_logic;  
        almost_full_inst  : out std_logic;  
        full_inst         : out std_logic;
```



```

        error_inst          : out std_logic;
        wr_addr_inst : out std_logic_vector(bit_width(inst_depth)-1
                                              downto 0);
        rd_addr_inst : out std_logic_vector(bit_width(inst_depth)-1
                                              downto 0) );
end DW_fifoc1_s1_sf_inst;

architecture inst of DW_fifoc1_s1_sf_inst is
begin

    -- Instance of DW_fifoc1_s1_sf
    U1 : DW_fifoc1_s1_sf
        generic map (depth => inst_depth,    ae_level => inst_ae_level,
                    af_level => inst_af_level,  err_mode => inst_err_mode,
                    rst_mode => inst_rst_mode )
        port map (clk => inst_clk,    rst_n => inst_rst_n,
                 push_req_n => inst_push_req_n,    pop_req_n => inst_pop_req_n,
                 diag_n => inst_diag_n,    we_n => we_n_inst,
                 empty => empty_inst,    almost_empty => almost_empty_inst,
                 half_full => half_full_inst,    almost_full => almost_full_inst,
                 full => full_inst,    error => error_inst,
                 wr_addr => wr_addr_inst,    rd_addr => rd_addr_inst );
end inst;

-- pragma translate_off
configuration DW_fifoc1_s1_sf_inst_cfg_inst of DW_fifoc1_s1_sf_inst is
    for inst
    end for; -- inst
end DW_fifoc1_s1_sf_inst_cfg_inst;
-- pragma translate_on

```

## HDL Usage Through Component Instantiation - Verilog

```
module DW_fifoc1l_s1_sf_inst(inst_clk, inst_rst_n, inst_push_req_n,
                             inst_pop_req_n, inst_diag_n, we_n_inst,
                             empty_inst, almost_empty_inst, half_full_inst,
                             almost_full_inst, full_inst, error_inst,
                             wr_addr_inst, rd_addr_inst );

    parameter depth = 4;
    parameter ae_level = 1;
    parameter af_level = 1;
    parameter err_mode = 0;
    parameter rst_mode = 0;
    `define bit_width_depth 2 // ceil(log2(depth))

    input inst_clk;
    input inst_rst_n;
    input inst_push_req_n;
    input inst_pop_req_n;
    input inst_diag_n;
    output we_n_inst;
    output empty_inst;
    output almost_empty_inst;
    output half_full_inst;
    output almost_full_inst;
    output full_inst;
    output error_inst;
    output [`bit_width_depth-1 : 0] wr_addr_inst;
    output [`bit_width_depth-1 : 0] rd_addr_inst;

    // Instance of DW_fifoc1l_s1_sf
    DW_fifoc1l_s1_sf #(depth, ae_level, af_level, err_mode, rst_mode)
        U1 (.clk(inst_clk), .rst_n(inst_rst_n),
            .push_req_n(inst_push_req_n), .pop_req_n(inst_pop_req_n),
            .diag_n(inst_diag_n), .we_n(we_n_inst), .empty(empty_inst),
            .almost_empty(almost_empty_inst), .half_full(half_full_inst),
            .almost_full(almost_full_inst), .full(full_inst),
            .error(error_inst), .wr_addr(wr_addr_inst),
            .rd_addr(rd_addr_inst) );
endmodule
```

## Revision History

For notes about this release, see the [DesignWare Building Block IP Release Notes](#).

For lists of both known and fixed issues for this component, refer to the [STAR report](#).

For a version of this datasheet with visible change bars, click [here](#).

Date	Release	Updates
July 2020	DWBB_201912.5	<ul style="list-style-type: none"><li>Adjusted content and title of “<a href="#">Suppressing Warning Messages During Verilog Simulation</a>” on page 9 and added the DW_SUPPRESS_WARN macro</li></ul>
October 2019	DWBB_201903.5	<ul style="list-style-type: none"><li>Added the “Disabling Clock Monitor Messages” section</li></ul>
March 2019	DWBB_201903.0	<ul style="list-style-type: none"><li>Clarified license requirements in <a href="#">Table 1-3</a> on page 3</li><li>Added “<a href="#">Enabling minPower</a>” on page 16</li></ul>
January 2019	DWBB_201806.5	<ul style="list-style-type: none"><li>Updated example in “<a href="#">HDL Usage Through Component Instantiation - VHDL</a>” on page 16</li><li>Added this Revision History table and the document links on this page</li></ul>

## Copyright Notice and Proprietary Information

© 2022 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

### Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

### Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

### Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>.

All other product or company names may be trademarks of their respective owners.

### Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

### Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.

[www.synopsys.com](http://www.synopsys.com)